Tuesday, May 16, 2017

Blindfind Visual Odometry Module Documentation

---

1. Overview

Visual Odometry module is written in c++ utilizing several open source libraries including opencv, g2o, lmmin, etc. It is aiming at solving the visual odometry problem, which given a video sequence in order, predicts the trajectory the camera travelled through. Currently this is only supporting a stereo visual odometry scheme, but providing flexible and adjustable APIs to handle monocular cases as well. The best way to approach it is to start from main.cpp and follow the functions it runs through. Major functions are documented here to help understand the code well. However, it is important to point out that neither the code, nor the documentation is well prepared. As a result, please always update the documentation, as the way it is now, whenever the code gets an upgrade.

All system parameters are recorded in the SystemParameter.h file. Please also do this in the future instead of putting things everywhere. When you see any variables, parameters, that are difficult to figure out the meaning of which, please calm down and turn to Rong Yuan at yuanrong0608@gmail.com.

2. Namespace

All functions in this project are under the namespace “blindfind”. If further reference is called for, please rememberer to specify the namespace, or get thousands of warnings.

3. Classes

Feature

The functions in this section are responsible for creating features and associating them with unique identifiers, a.k.a. feature-ids.

Data members

- cv::KeyPoint point2d
  Opencv type of keypoint.
- long id
  Unique identifier associated to the feature. See bindfind::IdGenrator for details in generation.

Member functions

- Feature(const cv::KeyPoint kp, const long id)
  Parameter:
  - kp
    Opencv KeyPoint object that defines the feature.
  - id
    Long int type identifier.
- cv::KeyPoint getPoint()
  Return value:
  - The Opencv KeyPoint object associated to the feature.
- long getId()
  Return value:
  - The id associated to the feature.

FeatureSet

A FeatureSet includes a set of features and an efficient way to accessing them. Functions in this section handles FeatureSet in terms of insertion, removal, and query.

Data members

- std::vector<cv::KeyPoint> featurePoints
  A vector of all key points in this feature set. Note that the order of this vector is in accordance to that of the vector of ids.
- std::vector<long> ids
  A vector of ids of all features. Note that the order of this vector is in accordance to that of the vector of feature points.
- std::map<long, int> idLoopup
  A loop-up table, in the structure of c++ std::map, that relates id to the index of the corresponding feature point in the feature set. This needs to be updated whenever the an update, including insertion, removal, etc, is performed on the feature set.

Member functions

- FeatureSet(const std::vector<cv::KeyPoint> featurePoints, const std::vector<long> ids)
  Parameter:
  - featurePoints
    KeyPoints in this feature set should be in the same order as ids.
  - Ids
    Ids in this feature set should be in the same order as KeyPoints.
- std::vector<cv::KeyPoint> getFeaturePoint()
  Return value:
  - The vector of feature points.
- std::vector<long> getIds()
  Return value:
  - The ids associated to the features.
- void setIds(std::vector<long>)
- void setFeaturePoints(std::vector<cv::KeyPoint>)
- blindfind::Feature getFeatureById(const long id) const
  Return value:
  - The feature associated with the given id is returned. Notice that if the id does not exist, a null object is returned instead. A better practice is to call blindfind::FeatureSet::hasId(long) before calling this function. See hasId(long) for more details.
- void addFeature(const blindfind::Feature feature)
  This function will update the data member idLookup.
- void addFeature(const cv::KeyPoint, const long id)
  This function will update the data member idLookup.

- void removeFeature(const long id)
  This function will update the data member idLookup.
  Parameter:
  - id
    The feature associated with this id will be removed.

- bool hasId(const long id)
  This function returns true if the given id exists in the feature set. Please call this function beforehand whenever retrieving a feature from the feature set with id.

- int size()
  This function returns the number of feature points within the feature set.

- void clear()
  This function removes all feature points and ids in the feature set.

View

A view is defined as follows:

- As camera moves along a trajectory in time, at every time sample the frame or frames of that time sample all together with the information associated to them defines a view.
- In particular, it contains the original images (one if monocular, two if stereo, depending on runtime configuration), feature sets (one for each image), pose of the first camera coordinate(left if stereo), time stamp (not in seconds, but in indices within the sequence), and indicators of stereo and keyView.
- A keyView is a special view. This is in accordance to the definition of a key frame in visual odometry. See BlindFind Visual Odometry Documents for details.

Member functions

- View(const std::vector<mat> imgs, int time)
  Parameter:
  - imgs
    A vector of cv::Mat images. Size 1 if monocular.
  - time
    The index, a timestamp, of the view.
- long getId()
- void setId()
- std::vector<cv::Mat> getImgs()
- void setImgs(const std::vector<cv::Mat> imgs)
- cv::Mat getPose()
- void setPose(cv::Mat)

- cv::Mat getR()
  Return value:
  - The 3x3 matrix that represents the rotation matrix of the pose. See introduction of class View for details.

- cv::Mat getT()
  Return value:
  - The 3x1 matrix that represents the translation vector of the pose. See introduction of class View for details.

- void setPose(const cv::Mat pose)

- void setPose(const cv::Mat R, const cv::Mat T)

- void setKeyView()

- void unsetKeyView()

- bool isKeyView()

- bool isStereo()

- int getTime()
  Return value:
  - This function returns the timestamp of this view. See introduction of class View for details.

Member functions

- ViewReader(const std::string dataset, const std::string track, bool stereo)
  Parameter:
  - dataset
    For example, make a copy of KITTI dataset under the specified image directory in blindfind::SystemParameters and write “KITTI” here.
  - track
    For example, “00”.
  - stereo
    If true, then at each time two images are read in. Otherwise one at a time.

- vector<Mat> next()
  It reads in next image or stereo images according to index and stereo indicator.

ViewReader

The view reader reads images from file. The dataset, track and indicator of stereo are required when reading. The configuration of file directories can be found in SystemParameters.

Data members

- std::string dataset
  The name of the dataset. ViewReader will locate this name under the given image files directory.
- std::string dataset
  The name of the track. ViewReader will locate this name under the given image files directory.
- bool::stereo
- int::index
  Index of the image to be read next. Whenever an image (or two if stereo) is read, this index increment by 1. User can specify a start index in blindfind::SystemParameters.

FeatureExtractor

Features are extracted in this class. The type of the feature is specified in SystemParameters. Note that this class is different than FeatureCandidateExtractor class in the fact that every feature extracted here will be assigned an unique id, while that from FeatureCandidateExtractor will not.

Member functions

- std::vector<cv::KeyPoint> extractFeatures(const cv::Mat img, blindfind::FeatureSet featureSet)
  New features, each of which associated with an unique identifier, are extracted. The type of the features to be extracted is specified in SystemParamters.
  Parameter:
  - img
    The image where features are extracted from.
  - featureSet
    Features extracted are assigned to featureSet.

  Return value:
    Key points extracted.

- std::vector<cv::KeyPoint> reextractFeatures(const cv::Mat img, blindfind::FeatureSet featureSet)
  This is different from blindfind::FeatureExtractor::extractFeatures. Existing features remain there while new features associated with new unique identifiers are added.
  Parameter:
  - img
    The image where features are extracted from.
  - featureSet
    Features extracted are assigned to featureSet.

  Return value:
    Key points extracted.

FeatureCandidateExtractor

Features are extracted in this class. The type of the feature is specified in SystemParameters. Note that this class is different than FeatureExtractor class in the fact that every feature extracted here will NOT be assigned an unique id, while that from FeatureExtractor will.

Member functions

- std::vector<cv::KeyPoint> extractFeatures(const cv::Mat img)
  New features, each of which NOT associated with an unique identifier, are extracted. The type of the features to be extracted is specified in SystemParamters.
  Parameter:
  - img
    The image where features are extracted from.

  Return value:
    Key points extracted.

- cv::Mat computeDescriptors(const cv::Mat img, std::vector<cv::KeyPoints> &kps)
  Descriptors of given key points on the given image are computed
  Parameter:
  - img
    The image where features are extracted from.
  - kps
    Key points whose descriptors are to be computed.

  Return value:
    NxM matrix, where N is the number of key points, and M is the dimension of a descriptor.

FeatureTracker

The functions in this section track features and apply Lowe's to refine.

Member functions

- kltTrack(const cv::Mat img1, const cv::Mat img2, const blindfind::FeatureSet featureSet1, blindfind::FeatureSet featureSet2)

  Parameter:
  - img1
    The image where features are tracked from.
  - img2
    The image where features are tracked to.
  - featureSet1
    The feature set where features are tracked from. Note that features in featureSet1 is not updated.
  - featureSet2
    The feature set where features are tracked to. Note that features in featureSet2 is updated.

- refineTrackedFeatures(const cv::Mat img1, const cv::Mat img2, const blindfind::FeatureSet featureSet1, blindfind::FeatureSet featureSet2)

  This function applies Lowe's on tracked features from KLT tracking. Therefore featureSet2 is assumed to contain tracked features. In particular, a search for feature match is performed in a small patch around the tracked feature. Here the tracked feature is considered as a prior. The best match, in terms of similarity of descriptors, is then taken as the feature match, if the distance of two descriptors has the value smaller than 70% of the distance of second best match in the patch. See BlindFind Document for more clarification.

  Parameter:
  - img1
    The image where features are tracked from.
  - img2
    The image where features are tracked to.
  - featureSet1
    The feature set where features are tracked from. Note that features in featureSet1 is not updated.
  - featureSet2
    The feature set where features are tracked to. Note that features in featureSet2 is updated. Different than kltTrack, here featureSet is assumed to contain tracked features to be used as priors.

- trackAndMatch(vector<View*> views)

  This is an interface that combines blindfind::FeatureTracker::kltTrack and blindfind::FeatureTracker::refineTrackedFeatures. The features are tracked from the first view in the vector of views, to the last view in the vector. Then, the features in the last view are refined. Note that features are assumed to be extracted in the first view in the vector.

  Parameter:
  - views

    Views where features are to be tracked through.

- trackAndMatch(View* view)

  This is an interface that combines blindfind::FeatureTracker::kltTrack and blindfind::FeatureTracker::refineTrackedFeatures. When the input is a single view rather than a vector of views, this function performs track and match between a stereo pair.

  Parameter:
  - view

    Features from left feature set in view are tracked and refined onto the right feature set.

PoseEstimator

Functions in this section estimate camera poses from feature correspondences.

Member functions

- cv::Mat estimatePoseMono(blindfind::View *v1, blindfind::View *v2)

  This function estimates the relative pose from v1 to v2 from given feature correspondences. Note that two feature sets, from v1 and v2, are not necessarily of the same size nor of the same order in terms of ids. In the implementation only matched features, correlated by ids, are used to compute relative pose. The core of this function is an opencv method cv::estimateEssentialMat. RANSAC, an outlier rejection scheme, is enabled during the estimation.

  * Note that nothing has been changed in this function. When estimating essential matrix, only inliers are considered in the estimation procedure, but they are not removed from the view.

  Return value:

  The relative pose from v1 to v2 is returned. It is a 4x4 global pose matrix consisting of a 3x3 rotation matrix R, and a 3x1 translation vector T, such that transferring a point P in current camera coordinate to that, P^w, in world coordinate, is through $P^w = RP + T$.

- double solveScalePnPRANSAC(blindfind::View *v, cv::Mat prevPose, std::map<long, blindfind::Landmark> landmarkBook, double initial = 1.0)

  This function estimates the scale of current translation vector T, with respect to the prevPose, of the given View v. Note that the returned scale is not necessarily the real scale of T. Instead, it is a scale factor multiple on current T such that certain objective function is optimized. For example, if the true translation vector T is given in the input View v, then the returned scale should be 1.
  In addition, this function does not change anything in the View v and landmark book. Only a predicted scale factor is returned.

  Parameter:

  - v

    The current View object with feature correspondences and a pose prior.
  - prevPose

    The previous pose in the form of a 4x4 cv::Mat matrix.
  - landmarkBook

    A map that stores all 3D landmarks, whose key is the feature id, and value is the corresponding 3D landmark.
  - initial

    The initial guess of the scale factor.

  Return value:

  The return value is a scale factor that is supposed to be multiplied to current T in order to minimize certain objective function. See function description for details.

- void solveScale(std::vector<blindfind::View> allViews, std::vector<blindfind::View> keyViews, bool trinocular)

  This function estimates the scale of current translation vector T, with respect to the prevPose, of the set of given key views, keyViews. All views are passed in as input as well so as to implement a backup scheme, which replaces some poorly-performing frames with nearby non-keyframes. User can specify the objective function be trinocular reprojection error by setting trinocular to true, or be the 3D reprojection error by setting it to false.
  After the function is run, poses of keyViews are updated. Besides, the candidates in keyViews might change as well due to poor performance.

  Parameter:
  - keyViews
    See function description.
  - allViews
    See function description.
  - trinocualr
    See function description.

  Return value:
    The return value is a scale factor that is supposed to be multiplied to current T in order to minimize certain objective function. See function description for details.

4. A Few More Words from the Annoying Author

Good luck.